ParallelNACluster: Uma estratégia paralela de clusterização para o casamento de múltiplos catálogos

Vinícius Pires de Moura Freire[1,3], **Fábio Porto**[2],
José A. F. de Macêdo [1]

[1]Departamento de Computação – Universidade Federal do Ceará (UFC)
Fortaleza – CE – Brasil

[2]Extreme Data Laboratory (DEXL Lab)
Laboratorio Nacional de Computacao Cientifica (LNCC)
Petrópolis – RJ – Brasil .

[3]Instituto Federal de Educação, Ciência e Tecnologia do Maranhão
Campus Coelho Neto – Maranhão – Brasil

vinicius.freire@ifma.edu.br, jose.macedo@lia.ufc.br, fabio@lncc.br

Abstract. *The astronomical catalogs cross-matching aims to identify common celestial objects present in different astronomical surveys. Traditional approaches in astronomy do not provide solutions to the problem of matching in the context of large data volume. In this paper, we have improved the NACluster algorithm by creating the ParallelNACluster strategy, a parallel version of NACluster that takes advantage of input data partitioning, and accepts large volumes of data even using a small hardware set. In addition, we propose the SCIBoundary, a new strategy for matching neighboring stars placed in different data partitions. The strategy leads to equivalent solutions in both NACluster and ParallelNACluster.*

Resumo. *O casamento de catálogos de astronomia tem o objetivo identificar objetos celestes em comum presentes em diferentes levantamentos astronômicos. As abordagens tradicionais em astronomia não apresentam soluções para o problema de casamento no contexto de grande volume de dados. Neste artigo, melhoramos o algoritmo NACluster ao apresentar a estratégia ParallelNACluster, uma versão paralela do NACluster que se aproveita do particionamento dos dados de entrada, e aceita grandes volumes de dados mesmo utilizando um conjunto de hardware de pequeno porte. Além disso, propomos o SCIBoundary, uma nova estratégia para tratamento do casamento de objetos espacialmente separados em partições de dados vizinhas. O SCIBoundary permite que obtenhamos resultados equivalentes entre o NACluster e o ParallelNACluster.*

1. Introdução

Levantamentos astronômicos usam instrumentos poderosos para varrer o céu e identificar objetos de interesse dentro da região pesquisada. Tais varreduras do céu dão origem ao mapeamento do céu, conhecido como catálogo. Os catálogos cobrem uma determinada faixa do céu e muitos deles possuem interseções em suas coberturas. Dessa forma, é interessante obter uma visão integrada deles, identificando os objetos em comum. Para isso, é necessário realizar o casamento de dados entre os catálogos, em outras

palavras, o *cross-matching* de catálogos. Em banco de dados, esse problema é conhecido como resolução de entidade. Em [Freire et al. 2014] apresentamos o NACluster, como solução para realizar o casamente de múltiplos catálogos. Essa solução, assim como as abordagens tradicionais funcionam de forma centralizada e são ineficientes para o casamento de grandes volumes de dados.

Usar grandes volumes de dados em algoritmos de casamento que funcionam de forma centralizada incorre em um tempo de execução extremamente alto ou incapacidade de executar por falta de memória. Em particular, para realizar o casamento entre bilhões de objetos através do NACluster de forma centralizada é necessário ter uma máquina com uma grande quantidade de memória RAM. Dessa forma, existe uma limitação física na maioria das máquinas usadas pela comunidade que impedem a realização de casamentos de grandes catálogos através do NACluster. Essas características nos motivam a criar uma versão distribuída do NACluster.

Este trabalho tem duas principais contribuições. A primeira contribuição é a proposta do algoritmo *ParallelNACluster*, uma solução distribuída para a resolução de entidades de vários catálogos espaciais através de técnicas de clusterização. O algoritmo tem o objetivo de reproduzir a clusterização do NACluster em um ambiente de larga escala com eficiência. De modo que, inicialmente, produza um particionamento dos dados de entrada tal que as fronteiras entre as partições sejam conhecidas e que haja um balanceamento das partições, ou seja, cada uma delas tenha um número semelhante de objetos e/ou variação de densidades semelhantes, para que o NACluster seja executado em cada partição com tempo relativamente próximo. O algoritmo paralelo mostrou-se eficiente e produziu clusters com qualidade semelhante ao centralizado, a diferença mínima apresentada foi em torno de 10^{-4} na precisão do resultado.

A segunda contribuição é a criação do *SCIBoundary*, uma estratégia para o tratamento das fronteiras geradas pelo particionamento e que objetiva encontrar os objetos influenciados por elas. Como o NACluster precisa da vizinhança para realizar o casamento, segundo seu algoritmo, e os dados particionados fazem com que a vizinhança da fronteira de duas partições vizinhas seja separada, é preciso conhecer as fronteiras e os objetos próximos em partições vizinhas para a que execução da clusterização seja feita corretamente sobre eles. Nos experimentos realizados constatamos a viabilidade do *SCIBoundary* e que a qualidade dos clusters produzidos por ele é semelhante aos do NACluster e ParallelNACluster, com uma diferença em torno de 10^{-2} para baixo no valor da precisão, justificada pelo tamanho da amostra ser bem menor que nos demais.

Este artigo está organizado da seguinte forma. Na seção 2 apresentamos o algoritmo centralizado NACluster, essencial para o entendimento da sua versão paralela proposta. A seção 3 apresenta as principais contribuições deste trabalho, o ParallelNACluster e o SCIBoundary. A seção 4 mostra os experimentos utilizados para validar as estratégias. Na seção 5 apresentamos os trabalhos relacionados. Finalmente, na seção 6 são feitas as considerações finais sobre este estudo sugerindo possíveis temas para trabalhos futuros.

2. Casamento de Múltiplos Catálogos

NACluster [Freire et al. 2014] é um acrônimo para **N-way CAtalog Clustering**, um algoritmo de clusterização para realizar casamentos entre múltiplos catálogos espaciais. Um conjunto S de *n* catálogos é fornecido como entrada e a saída é uma clusterização

composta por k clusters, onde todos os objetos associados ou mapeados a um cluster devam vir originalmente de catálogos diferentes e cada cluster deve representar um objeto espacial na região observada. Assim, não se sabe de antemão quantos clusters serão produzidos ao final da execução do algoritmo. Estas restrições nos abstém de adotar um algoritmo de clusterização tradicional, como *k-means*.

O NACluster é uma adaptação do *K-means*, onde os agrupamentos representam as entidades para as quais os objetos de catálogos devem ser mapeados. Assim, no NACluster, para cada objeto dos catálogos envolvidos, procura-se o cluster cuja distância entre seu centroide e o objeto seja menor que um valor dado, chamado de $epsilon$ (ϵ).

A ideia do algoritmo é comparar cada objeto com centroides de clusters distantes até ϵ, definido pelo usuário, usando a distância Euclidiana $d(o_i, C_a)$ de um objeto o_i para um centroide C_a. Quando existe um centroide C_a, tal que $d(o_i, C_a) < \epsilon$, então este centroide é classificado como potencial candidato a ser mapeado ao objeto o_i. Dentre os potenciais candidatos, aquele com menor distância ao objeto o_i é mapeado a ele. Esse mapeamento, no entanto, só pode ser aplicado quando não existe um outro objeto o_j no cluster de C_a que faça parte do mesmo catálogo de o_i.

No caso de um objeto o_j do mesmo catálogo de o_i já existir no cluster de C_a, é preciso tratar a colisão de mapeamento. Se $d(o_j, C_a) > d(o_i, C_a)$ então é necessário remover o objeto o_j do cluster de C_a, inserir o_i neste cluster, e procurar um outro cluster para o_j. Se $d(o_j, C_a) < d(o_i, C_a)$ então o algoritmo busca dentre os demais centroides candidatos a serem mapeados ao objeto o_i um novo cluster para realizar o mapeamento. Caso nenhum cluster seja encontrado na distância ϵ, um novo cluster com centroide C_b é criado para o ponto o_i, no qual sua posição é o centroide. O algoritmo termina quando todos os objetos de todos os catálogos forem avaliados.

A implementação do NACluster foi feita em Java está disponível à comunidade[1]. Para indexar os centroides usamos a *PH-tree*[Zaschke et al. 2014], reduzindo, assim, o espaço de busca. Os experimentos realizados são apresentados em [Freire et al. 2014].

3. Casamento Distribuído de Catálogos

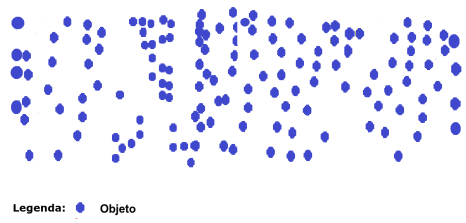
Conforme discutido anteriormente, este trabalho apresenta uma estratégia para a execução do casamento de catálogos astronômicos de forma distribuída e paralela. Neste sentido, considerando-se as técnicas atuais de processamento de grandes volumes de dados como o arcabouço Spark, existe a necessidade de particionar o conjunto de dados de entrada pelos nós de um cluster de computadores. Dessa forma, o conjunto de dados deve ser divididos em partes menores, chamadas partições. O particionamento distribui objetos pelos nós de processamento induzindo acesso distribuído quando referências a objetos em nós distintos são realizadas. A necessidade de particionamento vai de encontro com a noção de proximidade utilizada pelo método de clusterização. É importante que o aspecto físico de alocação de dados em partições não prejudique a noção lógica de proximidade entre os objetos, necessária ao correto processo de clusterização por vizinhança. De modo a tratar esse problema, introduz-se o conceito de fronteira que determina critérios lógicos de pertinência de objetos em partições. Considerando-se objetos espaciais identificados por sua posição no espaço 2D, as fronteiras formam um conjunto de posições em uma das dimensões que delimitam as partições dos dados.

[1]https://github.com/vinipires/NACluster

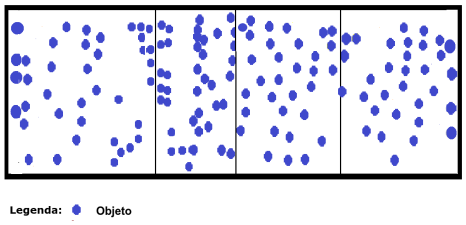
Nesta seção, estamos interessados em técnicas que permitam a paralelização do algoritmo NACluster. Suas características criam alguns desafios a serem superados na paralelização: 1 - os objetos vizinhos distantes em até ϵ devem ser executados juntos, portanto devem estar associados à mesma partição lógica; 2 - a alocação dos objetos em diferentes nós de forma desbalanceada pode fazer com que algumas máquinas executem o NACluster por mais tempo. Ou seja, enquanto as máquinas com mais objetos continuariam a clusterização, as demais terminariam o casamento e ficariam ociosas até o processo terminar por completo; 3 - o particionamento de um conjunto de dados gera fronteiras, f_i e f_{i+1}, entre partições, e essas fronteiras precisam ser conhecidas e tratadas.

3.1. Arquitetura Geral

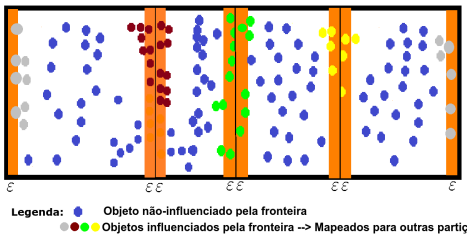
Neste artigo, propomos um modelo MapReduce para ser executado no framework de processamento distribuído Apache Spark[2]. Damos o nome de *ParallelNACluster* à solução proposta nos próximos parágrafos. Resumidamente, ela é dividida em quatro etapas principais ilustradas na Figura 1.



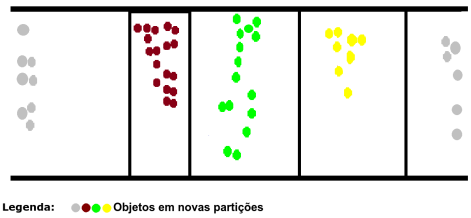
(a) Dados Originais



(b) Particionamento



(c) **Map** - NACluster aplicado em cada partição e os objetos influenciados pela fronteira são mapeados para uma nova partição e enviados ao Reduce



(d) Reduce - NACluster em cada nova partição

Figura 1. Etapas do Particionamento ao Reduce

Particionamento é fase em que os dados são particionados de acordo com os critérios apresentados anteriormente nos desafios da paralelização. O particionamento produz partições balanceadas e as fronteiras são conhecidas. A Figura 1(b) ilustra o particionamento a partir dos dados da Figura 1(a). **Map** é etapa em que executamos independentemente o NACluster em cada partição. Em seguida, identificamos os clusters que contêm objetos influenciados por cada fronteira, com o objetivo de mapear esses objetos para uma nova partição. A Figura 1(c) ilustra os não influenciados pela fronteira (os azuis) e influenciados pela fronteira (os demais). Mais detalhes serão apresentados na

[2]http://spark.apache.org/

seção 3.3. Os clusters não influenciados pela fronteira formados em cada partição, depois da execução do NACluster, são guardados para serem utilizados na última etapa (União). **Reduce** é o passo em que efetuamos o NACluster nas novas partições geradas, as quais contêm os objetos influenciados por cada fronteira. Esses objetos precisam ser executados juntos para obedecer os critérios de proximidade exigidos na clusterização do NACluster. A Figura 1(d) ilustra as novas partições e seus objetos. **União** é a fase em que unimos os clusters produzidos nas etapas Map e Reduce e geramos o resultado final.

3.2. Particionamento

Desejamos realizar um particionamento do *dataset* de maneira que os objetos vizinhos sejam associados à mesma partição lógica. Além disso, as partições precisam ser balanceadas, ou seja, cada partição deve ter um número semelhante de objetos.

A proposta do particionamento para este trabalho vem de [Gaspar and Porto 2014], onde é proposto o FRANCE (FRAgmeNtador de Catálogos Espaciais), um algoritmo iterativo para particionar dados em histogramas *equi-depth*. Formalizamos o particionamento através da Definição 1.

Definição 1. *(Particionamento) Dado um objeto* $o_{ji} < x, y, a_1, a_2, ..., a_t >$ *tal que* a_z, $1 \leq z \leq t$, *é um atributo de* o_{ji}, *e* $x \in X$ *e* $y \in Y$, *tal que* X *e* Y *são as dimensões espaciais; um particionamento* P *é uma lista de valores* $< v_1, v_2, ..., v_g >$ *tal que* $v_h \in x$ *e* $1 \leq h \leq g$. *Desta forma, uma partição* P_r, *tal que* $1 \leq r \leq g - 2$ *apresenta objetos vizinhos em uma região do espaço delimitada por* P_r *e* P_{r+1}.

O FRANCE calcula iterativamente os pontos de fragmentação segundo uma das dimensões espaciais, RA, se preocupando em manter as partições com uma quantidade equivalente de elementos (dado um δ), mesmo em catálogos com densidade não uniforme (como o Espaço).

As partições são geradas iterativamente até que a quantidade de elementos seja próxima em δ do balanceamento perfeito. Consideramos perfeito, balanceamentos em que as partições apresentam n/g elementos, onde n é o total de elementos e g a quantidade de partições.

O algoritmo tem como entrada o mesmo tipo de *dataset* usado no NACluster e gera como saída os valores da coordenada x onde ocorreram as divisões do espaço. A partir desses valores, podemos definir formalmente uma fronteira através da Definição 2.

Definição 2. *Uma fronteira* f_i *definida segundo um valor de particionamento* v_i *pertencente a* P, *contém um conjunto de objetos* O_{f_i} *cujos valores da coordenada* x *de particionamento estão entre* $v_i - \epsilon \leq x \leq v_i + \epsilon$.

Para maiores detalhes, a implementação do FRANCE que fizemos em java está disponível à comunidade na internet[3].

3.3. Tratamento de Fronteiras - Map (SCIBoundary), Reduce e União

Uma vez que as fronteiras tenham sido definidas, podem-se gerar as partições de dados a partir das associação de cada objeto do catálogo a um intervalo f_i e f_{i+1}. As partições assim criadas podem ser distribuídas pelos nós do cluster para processamento.

[3]http://github.com/vinipires/FRANCE

Assim, o NACluster pode ser executado em cada partição de forma isolada. A estratégia de processar isoladamente funciona para o NACluster, pois sua operação consiste em comparar cada objeto com possíveis centroides à uma distância ϵ. Portanto, é um processamento de vizinhança, e enquanto os vizinhos estiverem na mesma partição, o algoritmo tem um comportamento semelhante ao centralizado. A diferença aparece quando o vizinho está em outra partição.

O problema da fronteira pode ser enunciado da seguinte forma: dadas duas partições vizinhas P_i e P_j, uma fronteira f_k, $f_k \subset P_i$ e $f_k \subset P_j$, e dois objetos $o_i \in Pi$, e $o_j \in P_j$, se distância$(o_i, o_j) < \epsilon$ então o NACluster deve considerar f_k tal que $f_k \supset \{o_i, o_j\}$. Desta forma, para tratar o problema da fronteira, criamos a partir de P_i e P_j a fronteira f_k para processamento.

Como o particionamento do FRANCE é feito em apenas uma dimensão, existe somente uma fronteira entre duas partições vizinhas. Como a região onde pode aparecer ambiguidade na nossa definição de casamento está à uma distância ϵ do ponto referencial, e considerando que esse referencial é a fronteira, todos os elementos pertencentes à região distante ϵ dela precisam ser tratados nas duas partições.

Vamos expor a intuição de como podemos visualizar o tratamento das fronteiras. Essa visualização é mostrada através das componentes conexas de um grafo. Elas revelam existência de influência dos objetos da fronteira na formação dos clusters próximos à eles. As componentes conexas são quaisquer subgrafos conexos maximais de G, mais precisamente, são os maiores conjuntos de nós, tal que todos os nós conseguem alcançar os demais. Dado um grafo GC, formado a partir do resultado da execução do NACluster em cada partição, podemos observar todas as componentes conexas que têm origem na fronteira e detectar todos os clusters que tem influência da fronteira na sua formação. Caso processemos, juntos, todos os clusters que formam as componentes conexas que tem origem na fronteira de duas partições vizinhas em uma segunda etapa resolveremos o problema da fronteira.

Os nomes das partições que representam os objetos fora da fronteira são formados por Px, onde x corresponde ao número da partição. Os nomes das fronteiras são formados pelo nome da partição da esquerda unido com o nome daquela que fica à direita. Por exemplo, a fronteira que fica entre as partições $P1$ e $P2$ recebe o nome de $p1p2$.

O passo a passo do SCIBoundary é apresentado na Figura 2. Podemos examiná-la juntamente com os passos descritos acima. As elipses azuis representam os objetos, os retângulos vermelhos equivalem aos clusters (centroides) da fronteira e influenciados por ela, e os retângulos pretos correspondem aos clusters (centroides) fora da fronteira ou não influenciados por ela. Podemos observar que no momento "a" da Figura 2, possuímos os clusters (em preto) formados pelo NACluster, classificados inicialmente como fora da fronteira, e um objeto (em azul) pertencente à ela. Todos eles presentes na partição *P1*. No momento "b", o objeto procura por centroides de clusters em um raio ϵ e encontra dois deles. Estes dois clusters passam a ter a cor vermelha no instante "c", ou seja, passam a ser considerados como clusters influenciados pela fronteira e são mapeados para partição *p1p2*, e em "d" a componente conexa começa a se formar. No momento "e", os objetos do cluster encontrado são recuperados e inicia-se uma busca por outros clusters (centroides) fora da fronteira em "f". No instante "g", o novo cluster encontrado transforma-se em

cluster influenciado pela fronteira, tornando-se vermelho. De "h" até "n", os passos são semelhantes aos ocorridos anteriormente. Ao final da execução do SCIBoundary em "n", todos os clusters encontrados a partir da fronteira ficaram vermelhos e foram mapeados para a partição de mesmo nome da fronteira *p1p2*, mostrando que eles são influenciados por ela.

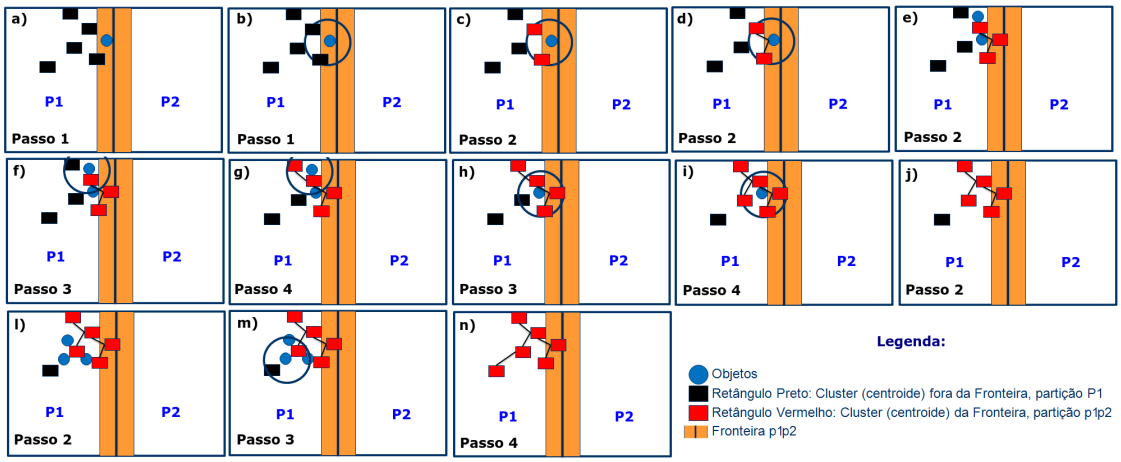


Figura 2. Passo a passo do SCIBoundary

No fim da execução do SCIBoundary em partições vizinhas teremos dois conjuntos de dados: o conjunto **A** contendo os clusters que não são afetados pela fronteira, e o conjunto **B** contendo os clusters afetados pela fronteira. Os clusters do conjunto **A** devem ser guardados para serem usados no resultado final e os clusters do conjunto **B** devem ser enviados para a etapa Reduce.

Um exemplo ilustrativo da classificação feita pelo SCIBoundary em todo o *dataset* é apresentado na Figura 3. Neste exemplo, o *dataset* possui 4 partições inicialmente. Ao final do SCIBoundary, o *dataset* passa a ter 8 partições, as 4 já existentes anteriormente ($P1$, $P2$,$P3$ e $P4$) e as 4 novas partições correspondentes às fronteiras ($p1p2$, $p2p3$, $p3p4$ e $p4p1$). A cor das elipses próximas às fronteiras representa a partição a qual os clusters pertencem.

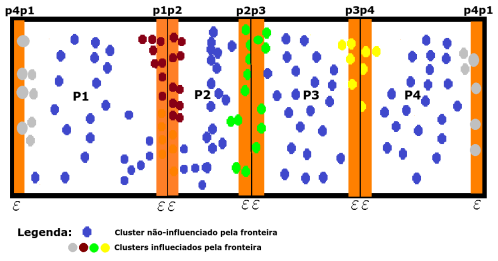


Figura 3. Exemplo ilustrativo dos clusters após a execução do SCIBoundary.

As novas partições criadas pelo SCIBoundary contendo os clusters influenciados pela fronteira são processadas na etapa Reduce. A Figura 4 ilustra as partições formadas. Cada elipse representa um cluster influenciado pela fronteira e sua cor representa sua partição composta por clusters que devem ser reexecutados, desta vez juntos na mesma

partição. O Reduce recebe como entrada o par (chave,valor), onde a chave é o *id* da nova partição e o valor é a lista dos clusters dessa partição. Para cada partição, são recuperados os objetos pertencentes à todos os seus clusters e é executado o NACluster sobre esses objetos, produzindo, assim, um novo casamento. Uma observação acerca desta etapa é que apesar dos objetos das fronteiras de *RA* 0.0 e 360.0 executarem juntos no Reduce, eles não estão sendo adaptados para corrigir seus clusters. A implementação foi feita dessa forma para facilitar o tratamento dessa fronteira em trabalhos futuros.

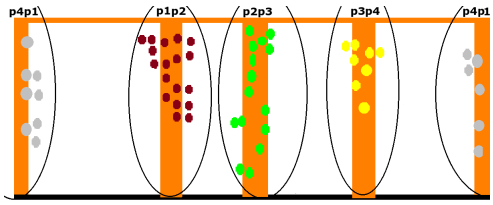


Figura 4. Partições formadas após a execução do SCIBoundary.

O resultado final do ParallelNACluster será a união dos clusters não influenciados pela fronteira, armazenados previamente na etapa *Map*, com os novos clusters produzidos pela etapa Reduce. A conclusão dessa etapa garante que o resultado do ParallelNACluster seja semelhante ao resultado do NACluster Centralizado aplicado ao mesmo *dataset*. A implementação do algoritmo paralelo está disponível à comunidade[4].

4. Experimentos

O conjunto de catálogos S dado como entrada no NACluster aparece implementado em um aquivo no formato texto. Cada linha contém uma tupla com a forma *(idObjeto, ra, dec, idCatalogo)*, cujos elementos são autoexplicativos. Quaisquer catálogos de dados espaciais que sejam de mesma natureza podem ser casados pelo NACluster. Utilizamos os catálogos de astronomia em todos os nossos experimentos.

Conforme visto na Seção 2, durante a execução do NACluster, para cada objeto dos catálogos envolvidos, procura-se o cluster cuja distância entre seu centroide e o objeto seja menor que um valor dado, chamado de ϵ. Além disso, cada cluster deve representar um objeto espacial na região observada. Para constatar que cada agrupamento formado pelo NACluster é um objeto espacial, precisa-se de um conjunto de catálogos S que possua um *Golden Standard* do seu casamento, ou seja, um conjunto de catálogos no qual saibamos previamente os casamentos corretos. No entanto, não encontramos nenhum conjunto de catálogos com essa característica, dificultando, assim, medirmos a qualidade do algoritmo. Portanto, geramos catálogos sintéticos e usamos os catálogos originais como *Golden Standard* com o propósito de avaliar a qualidade do casamento. Para criar um objeto celeste sintético, tomamos a posição do objeto real e adicionamos aleatoriamente um erro e, dentre os presentes na distribuição das distâncias de no máximo 1 arcseg entre os objetos de dois grandes catálogos, *2MASS* e *UCAC3*. Logo após, geramos aleatoriamente um ângulo θ (de 0 a 360 graus) ao qual esta nova posição estará deslocada em relação à original. Consideramos que objetos de catálogos diferentes distantes em raio de 1 arcseg (segundo de arco) são potenciais candidatosa ao casamento.

[4]https://github.com/vinipires/ParallelNACluster

#Catálogos	Tamanho	#Objetos	Precisão	Abrangência	Medida-F
3	31 GB	946 Milhões	0,99947	0,99947	0,99947
4	39 GB	1,1 Bilhão	0,99847	0,99849	0,99848
5	47 GB	1,4 Bilhão	0,99847	0,99849	0,99848
6	54 GB	1,6 Bilhão	0,99828	0,99831	0,99829

Tabela 1. *Métricas de qualidade do ParallelNACluster aplicado a diferentes conjuntos de catálogos baseados no 2MASS*

Partindo do pressuposto que o casamento destes dois catálogos esteja correto, assumimos que essas são distâncias entre o mesmo objeto presente nos dois catálogos e as usamos para construir o *Golden Standard*. Consideramos cada distância da distribuição como um erro na posição do mesmo objeto em diferentes catálogos. Assim, podemos gerar *n* catálogos sintéticos, a partir de um catálogo real e ter um *Golden Standard*, pois sabemos qual objeto real gerou um sintético, e um casamento entre eles deve ser considerado correto.

O hardware utilizado nos testes com o ParallelNACluster é um *cluster*[5] formado por 5 máquinas comuns presentes no *DEXL Lab* (laboratório pertencente ao Laboratório Nacional de Computação Científica): **Master**: Processador core i7-4790, com 4 núcleos e 8 *threads* de 3,60Ghz. Além de memória RAM de 16G; **slaves 1, 2 e 3**: Processador AMD Phenom(tm) II X2 B55 com 2 núcleos de 3,00 Ghz e 8GB de memória RAM; **slave 4**: Processador core i5-3470, com 4 núcleos de 3,20Ghz e memória RAM de 8G.

4.1. Qualidade na execução de grandes catálogos

A Tabela 1 apresenta as medidas de qualidade dos casamentos feitos através do ParallelNACluster quando aplicado a grandes conjuntos simulados de catálogos gerados a partir de todos os objetos do *2MASS* com tamanhos variados de 31 GB a 54 GB e particionados em 500 partições. Cada linha corresponde à uma execução em um conjunto de catálogos de tamanho diferente. A primeira coluna, *#Catálogos*, apresenta a quantidade de catálogos. A segunda coluna mostra o tamanho do arquivo e a terceira coluna, *#Objetos*, representa a quantidade aproximada de objetos. As próximas colunas, *Precisão*, *Abrangência* e *Medida-F*, cujos nomes são autoexplicativos, apresentam seus respectivos valores.

Observamos que as medidas de qualidade permanecem por volta de 0,99, mesmo aumentando a quantidade de catálogos presentes nos conjuntos. O ParalellNACluster produziu um resultado cuja a diferença média apresentada foi em torno de 10^{-4} de precisão se comparado ao algoritmo centralizado.

Também medimos a qualidade do casamento dos objetos influenciados pelas fronteiras, ou seja, aqueles identificados pelo *SCIBoundary* e casados na etapa *Reduce*. Como cada *dataset* possui 500 partições, identificamos 500 fronteiras. Nos conjuntos de catálogos testados, as fronteiras contêm aproximadamente $0,14\%$ do total de clusters. A Tabela 2 apresenta a qualidade do casamento ocorrido com os objetos da fronteira. A primeira coluna contém a quantidade de catálogos do *dataset*. A segunda mostra a quantidade total de clusters da fronteira. A terceira apresenta o número de agrupamentos da fronteira classificados de forma correta segundo o *Golden Standard*, enquanto a quarta coluna mostra a quantidade de clusters errados. Esses valores foram usados para calcular

[5]Cluster no sentido de ser formado por computadores ligados que trabalham em conjunto.

#Catálogos	#Clusters	#Clusters Corretos	#Clusters Errados	Precisão
3	646.361	613.845	32.516	0,9497
4	674.410	660.499	13.911	0,9794
5	692.301	659.987	32.314	0,9533
6	716.592	680.947	35.645	0,9503

Tabela 2. *Qualidade do casamento da fronteira na etapa Reduce dos mesmos conjuntos de catálogos da Tabela 1.*

a precisão, apresentada na coluna seguinte. O processo de clusterização realizado na etapa Reduce produziu um resultado cuja a diferença média apresentada foi em torno de 10^{-2} no valor da precisão se comparado ao NACluster e ParallelNAcluster, pois sua amostra é pequena, ou seja, a sua quantidade de *clusters* é muito menor que nos demais.

4.2. Escalabilidade: Volume x Tempo de execução

O objetivo dos próximos experimentos é avaliar a escalabilidade do ParallelNA-Cluster, ou seja, se ele está preparado para o crescimento do volume de dados. Usamos quatro diferentes conjuntos de catálogos com as seguintes quantidades aproximadas de objetos: 947 milhões, 1.18 Bilhão, 1.41 Bilhão e 1.65 Bilhão. E fixamos as seguintes configurações no Spark: 500 partições, –num-executors 4, –executor-cores 2, –excutor-memory 5G.

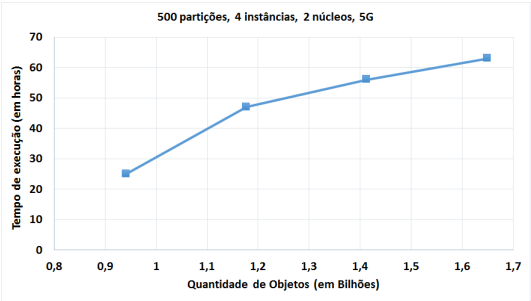


Figura 5. Volume x Tempo de uma Execução

Na Figura 5 apresentamos o tempo das execuções. No eixo x temos a quantidade de objetos e no eixo y o tempo de execução. Observamos que mesmo aumentando o tamanho do conjunto de catálogos para grandes volumes de dados, na ordem de bilhões de objetos, a aplicação executa por completo. É importante ressaltar que neste caso não temos a média do tempo, pois a aplicação demora de um a dois dias para executar grandes volumes de dados em nosso pequeno cluster, e problemas como energia e rede, impossibilitaram realizar dez rodadas para cada conjunto de catálogos.

Concluímos que mesmo com grandes volumes de dados é possível encontrar um equilíbrio entre o número de partições, o volume de dados e a memória disponível para cada instância. Logo, podemos crescer o tamanho do conjunto de catálogos contanto que ajustemos os parâmetros do Spark para um valor adequado ao tamanho dos dados. Em nossos experimentos encontramos uma escalabilidade linear até 1,7 bilhão de objetos em um cluster barato formado por apenas 4 máquinas.

5. Trabalhos Relacionados

Em [Dai and Lin 2012] foi proposta a estratégia DBSCAN-MR, uma abordagem que melhora a escalabilidade do DBSCAN dividindo os dados de entrada em partes menores e enviando-os aos nós para processamento paralelo. Em [Kwon et al. 2010] foi apresentado o algoritmo dFoF (*distributed Friends-of-Friends*, *amigos-de-amigos* distribuído, em português). Ele possui estratégia semelhante ao SCIBoundary e é a versão paralela do *FoF (Friends of Friends)*, um algoritmo de clusterização, semelhante ao DBSCAN, que aceita uma lista de objetos (*pid, x, y, z*) como entrada e retorna uma lista de tuplas de clusters (*pid, ClusterID*). Para calcular os agrupamentos, o algoritmo examina apenas a distância entre os objetos. *FoF* define dois objetos como amigos, se a distância entre eles é inferior a ϵ. No algoritmo $dFoF$ os dados são primeiramente particionados, depois agrupados localmente e, finalmente, os clusters locais são mesclados aos das partições vizinhas através do processamento da fronteira. Esses trabalhos relacionados propõem criar algoritmos de clusterização que funcionam de forma distribuída e dão ênfase ao particionamento dos dados de entrada, bem como do tratamento das fronteiras. No entanto, existem algumas diferenças quanto à natureza dos dados e, consequentemente, o objetivo final dos algoritmos apresentados nesta seção é diferente do objetivo da nossa proposta. Enquanto os algoritmos aqui apresentados estão interessados em unir os objetos próximos para identificar os grupos com formas arbitrárias, nós estamos interessados em formar agrupamentos de objetos próximos, mas que pertençam a diferentes catálogos.

Em [Zhao et al. 2009] apresentou-se o Parallel k-means, um algoritmo de agrupamento que classifica um conjunto de elementos de entrada em k clusters de forma paralela através de uma abordagem Map Reduce do k-means. A estratégia de casamento do ParallelNACluster é semelhante ao mapeamento de objetos a grupamentos no Parallel k-means em alguns aspectos como, por exemplo, no cálculo do centroide do cluster, no qual a posição de cada centroide é a média da posição dos objetos do cluster e ambos utilizam a abordagem de alocar o objeto ao cluster de centroide mais próximo. No entanto, apesar de existirem características semelhantes entre as duas abordagens, existem várias diferenças. Dentre elas o relaxamento do k, pois diferentemente do Parallel k-means, o número de clusters formados, k, não é dado como entrada no ParalllelNACluster e a quantidade final de clusters só é descoberta ao final da execução. Além disso, no Parallel K-Means os centroides estão dispostos globalmente, enquanto que no ParallelNAClusters, os centroides são controlados localmente. Outra diferença é que no ParallelNAcluster os centroides iniciais são pré definidos e o resultado é determinístico na maioria dos casos, enquanto que no Parallel k-means, os centroides iniciais são aleatórios e a sequência na formação dos clusters é geralmente diferente a cada execução.

Ainda sobre as diferenças entre os dois algoritmos, no ParallelNACluster, o número de objetos mapeados para um agrupamento não pode ultrapassar a quantidade de catálogos existentes. Cada cluster pode conter apenas um objeto de cada catálogo. No Parallel k-means não existe restrição quanto ao tamanho do cluster e o tipo de catálogo que ele pode alocar. Adicionalmente, no ParallelNACluster é possível que um objeto seja removido do cluster ao qual ele tenha sido adicionado anteriormente e seja alocado em outro cluster caso nele ocorra uma colisão de objetos do mesmo catálogo, ou seja, um objeto tentar ser adicionado em um cluster onde já existe outro objeto do mesmo catálogo. Neste caso, aquele que for mais próximo do centroide permanece no cluster. No Parallel k-means não existe tratamento de colisão e objetos de diferentes catálogos podem

fazer parte do mesmo cluster. Com tantas diferenças, podemos concluir que o Parallel k-means não iria satisfazer os pré-requisitos de casamento do ParallelNACluster. A falta das restrições no Parallel k-means, bem como o k fixo, fariam com que um cluster não representasse um objeto real.

6. Conclusão

O ParallelNACluster é uma solução distribuída que pode ser utilizada por toda a comunidade através do uso de máquinas baratas. O particionamento permite que sejam utilizados *datasets* de tamanho na ordem de bilhões de objetos, tornando nossa proposta uma solução atual e robusta, que funciona mesmo com o grande aumento do volume de dados, bastando apenas aumentar o número de partições, utilizando o mesmo *cluster*, à medida que aumenta o tamanho dos dados. Dentre os trabalhos futuros propomos realizar experimentos com o ParallelNACluster em hardwares de alto desempenho, adicionar na função de distância outros atributos com o intuito de melhorar a qualidade do casamento nas regiões ambíguas, criar uma estratégia para calcular as probabilidades de casamento e desenvolver um método para tratar casamentos de catálogos com escalas diferentes e permitir múltiplos *matchings*;

7. Agradecimentos

Este trabalho recebeu o apoio financeiro da CAPES (bolsa de doutorado). Fabio Porto foi parcialmente financiado pela Bolsa CNPq de produtividade processo: 310109/2015-9.

Referências

Dai, B.-R. and Lin, I.-C. (2012). Efficient Map/Reduce-Based DBSCAN Algorithm with Optimized Data Partition. In *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, pages 59–66, Washington, DC, USA. IEEE Computer Society.

Freire, V. P., Porto, F., Akbarinia, R., and de Macêdo, J. A. F. (2014). NACluster: A Non-supervised Clustering Algorithm for Matching Multi Catalogues. In *2014 IEEE 10th International Conference on e-Science*, pages 83–86. IEEE.

Gaspar, D. and Porto, F. (2014). A Multi-Dimensional Equi-Depth Partitioning Strategy for Astronomy Catalog Data.

Kwon, Y., Nunley, D., Gardner, J. P., Balazinska, M., Howe, B., and Loebman, S. (2010). Scalable clustering algorithm for N-body simulations in a shared-nothing cluster. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6187 LNCS:132–150.

Zaschke, T., Zimmerli, C., and Norrie, M. C. (2014). The PH-tree: A Space-efficient Storage Structure and Multi-dimensional Index. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 397–408, New York, NY, USA. ACM.

Zhao, W., Ma, H., and He, Q. (2009). Parallel k-means clustering based on mapreduce. In *IEEE International Conference on Cloud Computing*, pages 674–679. Springer.